Using the Digiac-3080 Simulator

The module `digiac.py` provides an API for executing Digiac-3080 instructions. `sim3080.py` calls that API and provides a command-line interface for the simulator. This documents those two programs, focusing on the user interface and lower-level API.

Table of Contents

Digiac-3080 Machine State

The simulator implements these aspects of the Digiac-3080.

- 12-bit program counter
- 4096 word memory, 25 bits per word; words are composed of a sign bit followed by 24 bits of magnitude.
- “A” and “B” accumulators each contains one word
- Pending input (not yet read from the paper tape or console teleprinter).
- Output to paper tape or the console teleprinter
- Changes to the above state items due to execution of the digiac instructions.

Inputs and Outputs to the machine use I/O to a local file for tape and cards. For the console terminal, standard input (stdin) and standard output (stdout) of the simulator process is used.

The sim3080.py Command Line Interface

Starting and Stopping the Simulator

Enter Virtual Environment and Start the Simulator

Use these commands to start the simulator running.

```
cd digiac-3080
source venv/bin/activate
python sim3080.py
Dg>
```

Exit from the Simulator

The `q` command, the `quit` command or an `End-Of-File` on standard input will cause the simulator to exit. For example:

```
Dg> quit
```

Examining and Modifying Machine State

The `e <item>` or `examine <item>` command may be used to display the contents of a register or a memory address in octal and as characters. This command requires an argument of what item is to be examined, a lowercase register name without quotes, “pc”, “a”, “b”, or an octal address not exceeding 7777.

The `d <item> <value>` or `deposit <item> <value>` command may be used to store a new value into a register or a memory address. This command requires two arguments. The first argument is the destination, a register name or address. The second argument is the octal value to be stored.

Examples of `examine` and `deposit`:

```
Dg> deposit pc 543
Dg> examine pc
PC: 0543
Dg> d 123 -6543210
Dg> e 123
0123: -06543210 6YJ8
```

The `status` command displays the overall CPU status. This includes PC, word at the address in the PC, registers, count of instructions executed, throttle speed, breakpoints and address compare stops. For example:

```
Dg> status
Digiac< PC: 1450->15000055 A: +00000002 B: +00000000 Icnt: 15328
IPS: 60 bpt:1450 acs:3671>
```

Running Digiac Programs

The `go` and `step` commands are used to begin the execution of digiac instructions. Execution will continue until stopped by one of the “stop conditions” discussed below.

When instruction execution stops, the simulator prints a message showing state typically from the last instruction executed. This state is composed of four fields:

1. **Icnt**: count of instructions that have been executed as a decimal integer,
2. **Old PC**: address of the last instruction, terminated by “: ”,
3. **Instr**: the last instruction, terminated by “ .. ”,
4. **Result**: a string indicating the side effect of the instruction.

For example, when a HLT instruction stops the CPU the state prints like this.

```
185  0034: 00000000 .. HALTED at 0035
```

The result string indicates that the PC contains 0035 after the CPU stopped. Here the string is repeated with the fields labeled.

```
+----+------+--------+  +----------------
|Icnt|Old_PC| Instr  |  | Result string ....
+----+------+--------+  +----------------
  185  0034: 00000000 .. HALTED at 0035
```

There is a tracing mode that can be enabled. If tracing is enabled, simulator state will get printed after every instruction is executed.

Stop Conditions

These are the situations where execution of digiac instructions will stop. Typical messages are shown with each condition.

- HLT instruction executed

```
  185  0034: 00000000 .. HALTED at 0035
```

- Executing an invalid or unimplemented instruction

```
  187  0035: 76543210 .. Invalid or Unknown OPCODE 76543210 at
0035
```

- ZeroDivisionError during a DIV instruction

```
  189  5000: 24401234 .. Divide by Zero Stop
```

- Read Tape instruction with no input file attached to the tape reader

```
  191  5000: 60006000 .. No Tape in PTReader
```

- Unexpected input byte value read by the Read Tape instruction. An extra message is printed at time of the detection with the bad byte value and decimal position in bytes of the input file.

```
Unexpected PT character = 0x44 at offset 34
  193  5000: 60006000 .. next addr:     6000
```

- Requested number of instructions from a `step` command has been executed An extra message with the decimal count of steps is printed if the requested number of steps was more than one.

```
Instruction count 5 reached
   11  5000: 10605000 .. A      <- +00000043.
```

- About to execute the instruction at a Breakpoint address.

```
   17  5003: 44005000 .. Breakpoint at 5003
```

- Reading or writing data from one of the Address Compare Stop addresses. These stops occur for load/store instructions and for instruction fetch by the CPU. Here are two examples

```
Read Memory address Compare Stop @ 5000
   21  4001: 14005000 .. A      <- +10605000
Dg> step
Write Memory address Compare Stop @ 5000
   22  4002: 30005000 .. [5000] <- +10605000
```

- Control-C at standard input or SIGINT received by the simulator process

```
 3658  4000: 10000000 .. Control-C
```

Generally, when execution stops, the PC has been incremented and now it points to the next instruction. A Breakpoint is an exception to this rule because execution is stopped before fetching the instruction.

Starting Digiac Programs

The `g [<addr>]` or `go [<addr>]` command is used to start executing digiac instructions. Execution begins at the address in the PC. If an optional octal address argument is supplied, the PC is set to that address before beginning execution. Execution continues until a new stop condition occurs. A message is printed when execution stops. For example:

```
Dg> go 1400
 15274  3327: 00000252 ..  HALTED at 3330
```

Stepping Digiac Programs

The `s [<num_instr>]` or `step [<num_instr>]` command is used to execute a limited number of instructions. By default, only one instruction will be executed. If an optional number is supplied, that is the maximum number of instructions to be executed as a decimal integer. Execution will stop earlier if a new stop condition occurs. A message is printed when execution stops. For example:

```
Dg> e pc
PC: 1433
Dg> step
 15358  1433: 04000054 ..  A      <- +00000020
Dg> s 10
Instruction count 10 reached
 15368  1436: 45001442 ..  no branch
Dg> e pc
PC: 1437
```

Using Breakpoints

A breakpoint is the address of an instruction where execution should stop. There may be any number of breakpoints. When the simulator is about to fetch the instruction from a breakpoint, it will stop execution of digiac instructions. The PC is not incremented and the instruction at that address has not been executed.

The `break [<addr>]` command is used to create or list breakpoints. When the optional octal address is specified, a breakpoint will be set at that address. If no address is given, all the present breakpoints will be printed out.

The `clear <addr>` command is used to delete a breakpoint at the given address.

When execution s started by a `go` or a `step` command, a breakpoint at the initial PC is ignored for the duration of one instruction. This makes it more convenient to proceed from a breakpoint. We see this occur when the breakpoint at 6000 is initially ignored as execution is started by the `go 6000` command.

```
Dg> break 6000
Dg> break 6003
Dg> break
Breakpoints: 6000 6003
Dg> go 6000
 3678  6003: 30005777 .. Breakpoint at 6003
Dg> status
Digiac< PC: 6003->30005777 A: -40120000 B: +00000000 Icnt: 3678
IPS: 60 bpt:6000:6003>
Dg> go
 3680  6000: 30012345 .. Breakpoint at 6000
Dg> clear 6000
Dg> break
Breakpoints: 6003
```

Address Compare Stops

An Address Compare Stop (ACS) is a memory address that is being monitored. Any reading or writing that address will print a message and raise a stop condition. The current digiac instruction is allowed to complete. ACS will trigger and a message prints out for any of these types of memory access –

- instruction fetch,
- load/store arithmetic operand,
- I/O instruction memory access,
- fetch an instruction for simulator state / trace messages, and
- simulator `examine` or `deposit` command

The `acstop [<addr>]` command is used to create or list Address Compare Stops. When the optional octal address is specified, an Address Compare Stop will be set at that address. If no address is given, all the present Address Compare Stops will be printed out.

The `aclear [<addr>]` command is used to delete an Address Compare Stop at the given address.

```
Dg> acstop
Address Compare Stops: None
Dg> d 1000 10001000
Dg> d 1001 30001002
Dg> acstop 1002
Dg> go 1000
Write Memory address Compare Stop @ 1002
    2  1001: 30001002 .. [1002] <- +10001000
Dg> go
Read Memory address Compare Stop @ 1002
Read Memory address Compare Stop @ 1002
    3  1002: 10001000 .. A      <- +10001000
Dg> aclear 1002
```

Throttling

The emulated digiac CPU is by default limited to a historically realistic speed of 60 instructions per second (IPS). The speed is set or examined by the `throttle [<value>]` command. If present the value must be a non-negative decimal integer. If the value is set to zero, the speed is not limited.

```
Dg> throttle
60 Instr/sec
Dg> throttle 0
Dg> throttle
not throttled
```

Tracing Digiac Instruction Execution

The simulator has one byte of trace flags that may be set or examined by the `trace [<value>]` command. If the value is present, the trace flags are set; otherwise they are displayed. When the low-order bit is set, the simulator will print out the machine state for every instruction executed. This shows much detail of the executing program.

```
Dg> trace
trace flags: 00h
Dg> trace 1
Dg> step 8
    6  0000: 60000000 .. next addr:     0100
    7  0001: 10000000 .. A      <- +60000000
    8  0002: 14610001 .. A      <- +60000100
    9  0003: 30000000 .. [0000] <- +60000100
   10  0004: 15000037 .. A      <- -00003200
   11  0005: 45000000 .. PC     <-      0000
   12  0000: 60000100 .. next addr:     0200
   13  0001: 10000000 .. A      <- +60000100
Instruction count 8 reached
```

Connecting to Data Files

The `attach <device> <file>` command is used to connect a binary data file to an emulated digiac device. The `detach <device>` command breaks the connection and closes the file.

The `<device>` must be one of the supported devices. At present only the paper tape reader, "`ptr`", is supported.

`<file>` is the path to a file visible in the local filesystem.

```
Dg> attach ptr tape/stok.ptp
Dg> detach ptr
```

Getting Help

The `help` command can be used to get brief usage messages for the CLI command set.

```
Dg> help help
List available commands with "help" or detailed help with "help
cmd".
Dg> help go
Start or continue instruction execution: GO [addr]
Dg> help break
Set breakpoint at addr: BREAK [1234]
```

Debugging the Simulator

The Python pdb debugger may be activated to debug the digiac simulator.

```
Dg> pdb
--Return--
> /home/rne/digiac-3080/sim3080.py(41)do_pdb()->None
-> set_trace()
(Pdb) break self.do_go
Breakpoint 1 at /home/rne/digiac-3080/sim3080.py:213
(Pdb) cont
Dg> go 1000
> /home/rne/digiac-3080/sim3080.py(215)do_go()
-> args = arg.split()
(Pdb) next
> /home/rne/digiac-3080/sim3080.py(216)do_go()
-> if len(args):
(Pdb)
```

Loading and Running the Stock Market Game

There is no dedicated facility for the simulator to load or dump memory contents. However a very small bootstrap loader program can be deposited into memory to read media.

The Stock Market game works in this way. A one-instruction program to read 64 words from tape into memory at address zero is used. Data read by this instruction would supply the rest of the loader. The loader automatically runs to read the game into memory. Finally the loader prints “READY” and halts after all of the tape has been read.

```
Dg> attach ptr tape/stok.ptp
Dg> deposit 0000 60000000
Dg> go 0000
READY
  185  0034: 00000000 .. HALTED at 0035
```

To play the game we need to know that the program start / restart address is 1400.

```
Dg> go 1400

                              WELCOME TO THE 3080 BIG BOARD.

TYPE EACH REPLY AND SPACE UNTIL THE CARRIAGE RETURNS.

DO YOU KNOW HOW TO PLAY?
```

The Digiac3080 API

The module `digiac.py` provides an API for executing Digiac-3080 instructions. The API is encapsulated in the class `Digiac3080`. This chapter documents the public interface to that class.

25-bit digiac words have two representations. The simulated registers use a tuple of (sign, magnitude) to store a one-bit sign and an unsigned 24-bit magnitude. Simulated memory is an array of 32-bit integers; magnitude in the low-order 24 bits and sign in the next bit, leaving seven high order bits unused.

Creating an emulated Digiac-3080

When the `Digiac3080` class is instantiated, machine state with random data in memory is created. These python statements create an emulated machine `d`.

```
from digiac import Digiac3080
d = Digiac3080()
```

The object `d` exposes the machine state via attributes and properties. It provides functions for referencing memory and for executing instructions.

Attributes of Digiac3080

The module provides these public attributes:

`d.a` A register content as a tuple of (sign, magnitude).

`d.b` B register content as a tuple of (sign, magnitude).

`d.acs` list of Address Compare Stop addresses.

`d.bpt` list of breakpoints addresses.

`d.instruction_count`
the number of instructions that have been executed.

`d.ips` the number of instructions per second. If this value is non-zero, the emulator will sleep **`(1/value)`** seconds for every digiac instruction executed.

`d.pc` the program counter, a 12-bit unsigned integer.

`d.ptp` the file object for the paper tape punch. The caller must handle **`attach`** and **`detach`**, respectively, by opening a file and setting this attribute to the file object or to **`None`** when no file is attached.

`d.ptr` the file object for the paper tape reader. The caller must handle **`attach`** and **`detach`**, respectively, by opening a file and setting this attribute to the file object or to **`None`** when no file is attached.

`d.run` this attribute advises the caller whether the CPU should stop executing digiac instructions. The instruction emulator will set this attrubute to **`False`** if a stop condition occurs. A caller should set **`d.run`** to **`True`** before calling **`d.exec()`** to indicate that any prior stop condition has been handled.

Properties of Digiac3080

These properties are available:

d.areg_string The value is a formatted string to display the contents of the A register showing register name, sign and magnitude. For example:
"A: +00000002"

d.breg_string The value is a formatted string to display the contents of the B register showing register name, sign and magnitude.

Functions of Digiac3080

These functions are provided by the emulator:

d.rm(addr) Reads memory and returns the digiac word from the **addr** address in the low-order 25 bits. An ACS stop condition occurs if **addr** is a member of the **d.acs** list.

d.wm(addr,val) Writes the 32-bit value **val** into the memory at the **addr** address. An ACS stop condition occurs if **addr** is a member of the **d.acs** list.

d.exec() Execute the digiac instruction at the address **d.pc** and return the result string. If a stop condition occurs, **d.run** will be set to **False**. Unless interrupted by a breakpoint or Control-C, **d.exec()** does these actions:

- Check if the value in **d.pc** is a member of the **d.bpt** list, if true, return the result string "**Breakpoint at <addr>**".
- If **d.ips** is non-zero, sleep for 1/**d.ips** seconds.
- Fetch the instruction to be executed.
- Increment **d.pc**.
- Emulate/execute the instruction and update machine state.
- Return the result string from instruction execution.

Note: Control-C and signals are not handled by this function. The caller should catch the corresponding python exception. Refer to function **run_virtual_machine()** in **sim3080.py** for an example.

Details of Instruction Emulation

HLT (Halt) Instruction

This instruction sets **d.run** to **False**. The result string is "**HALTED at <addr>**".

AND, CLA, CLS, ADD, SUB (Arithmetic/Load) Instructions

The result string displays the new value in the A register "**A <- ±<value>**".

MLT (Multiply) Instruction

Product of the integer in A and the integer memory operand is written to the A and B registers, most significant bits in A. The result string displays the new values in both of the registers "`AB: ±<value> <value>`".

DIV (Divide) Instruction

The integer in A is divided by the memory operand which has a radix point to the left of the most significant bit. Alternatively, one can think of this as an integer division of `( A * 2`^{24}`) / (<integer_memory_operand>)`. The remainder is placed into A and the quotient is placed into B. The result string displays the new values in both of the registers "`AB: ±<value> <value>`".

Note: Division by zero will raise a stop condition without changing the content of A or B. This likely is inaccurate compared to the behavior of a real Digiac-3080. When this occurs, result string is "`Divide by Zero Stop`".

STA, STB (Store) Instructions

The result string shows what was written to memory "`[<addr>] <- ±<value>`".

JMP, BR-, BR+, BRZ (Jump and Branch) Instructions

The result string depends upon whether a branch was taken. It will be either "`no branch`", or "`PC <- <addr>`".

TA, RT, TI (I/O) Instructions

These instructions transfer words between memory and an external I/O device, starting at the address contained within the instruction. This address is incremented after each word is written to memory. The result string displays the next address that would be written "`next addr: <addr>`".

Not Implemented Instructions

If an unimplemented opcode is encountered, the result string displays the bad instruction and its address "`Invalid or Unknown OPCODE <instruction_word> at <addr>`".

At the time of this writing, these instructions have not been implemented:

50_8 Type Octal, 62_8 Read Card, 64_8 Punch Tape.